

Intel® Itanium® Processor-Specific Optimization

In general, -O3, IPO and/or PGO, and utilizing the optimization reports (in the *Fine-Tuning* section) to control aliasing and improve memory utilization, provide the best performance for Itanium-based systems.

Windows® Command	Linux® Command	Comment
/G1	-tpp1	Targets optimization for the Itanium® processor.
/G2	-tpp2	Targets optimization for the Itanium 2 processor. Generated code is also compatible with the Itanium processor (default).
/QIPF_fma[-]	-IPF_fma[-]	Enables [disables] the combining of floating-point multiply operations and add/subtract operations.
/QIPF_fp_speculationmode	-IPF_fp_speculationmode	Enables floating-point speculations with one of the following modes: fast - Speculate floating-point operations. off - Disables speculation of floating-point operations. safe - Speculate only when safe. strict - This is the same as specifying off.
/Qauto_ilp32	-auto_ilp32	Specifies that the application cannot exceed a 32-bit address space, which allows the compiler to use 32-bit pointers whenever possible. To use this option, you must also specify -ipo[value]. Using the -auto_ilp32 option on programs that can exceed 32-bit address space (2**32) may cause unpredictable results during program execution. This option has no effect on systems with Intel® EM64T unless the -axP or -xP option is also used.
/Qftz[-]	-ftz[-]	Flushes denormal results to zero. The option is turned ON with -O3 by default. This only impacts the application when the main program or dll main is compiled.
/Qivdep_parallel	-ivdep_parallel	Indicates there is absolutely no loop-carried memory dependency in the loop where the IVDEP directive is specified.
/QIPF_fitacc[-]	-IPF_fitacc[-]	Enables [disables] optimizations that affect floating-point accuracy.
/QIPF_fit_eval_method{0 2}	-IPF_fit_eval_method0	Evaluates floating-point operands to the precision indicated by the program.

Fine-Tuning

Once you have identified performance hot-spots, you may need to provide the compiler with more information to fine-tune specific functions. The optimization and vectorization reports may show places where loops could not be optimized fully due to pointer aliasing or memory access overlaps, for example. Also, the *Intel® C++ and Fortran Compilers User's Guides* include details on other #pragmas, directives, and intrinsics that can be used to control software-pipelining, unrolling, vectorization, and prefetching for further fine-tuning within your application code.

Windows® Command	Linux® Command	Comment
/Qunroll[n]	-unroll[n]	Sets the maximum number of times to unroll loops. -unroll0 disables loop unrolling. The default is -unroll, which uses default heuristics.
/Qrestrict[-]	-[no]restrict	Enables [disables] pointer disambiguation with the restrict qualifier.
	-falias	Assumes aliasing in the program. (C++ Linux only)
	-ffnalias	Assumes aliasing within functions. (C++ Linux only)
/Oa	-fno-alias	Assumes no aliasing in program.
/Ow	-fno-fnalias	Assumes no aliasing within functions, but assumes aliasing across calls.
/Qalias_args[-]	-alias_args[-]	Implies arguments may be aliased [not aliased].
/Qopt_report	-opt_report	Generates an optimization report directed to stderr.
/Qopt_report_filefilename	-opt_report_filefilename	Specifies the filename for the optimization report.
/Qopt_report_levellevel	-opt_report_levellevel	Specifies the verbosity level of the output. Valid arguments are min (default), med , max .
/Qopt_report_phase name	-opt_report_phase name	Specifies the compilation name for which reports are generated. The option can be used multiple times in the same compilation to get output from multiple phases. Valid name arguments: ipo : Interprocedural Optimizer hlo : High Level Optimizer ilo : Intermediate Language Scalar Optimizer ecg : Code Generator omp : OpenMP® all : All phases
/Qopt_report_routine [rtn]	-opt_report_routine [rtn]	Specifies a routine rtn . Generates reports from all routines with names that include rtn as part of the name.
		By default, generates reports for all routines.
/Qopt_report_help	-opt_report_help	Displays all possible settings for -opt_report_phase . No compilation is performed.

Parallel Performance

The following options allow the compiler to help you parallelize your application for multi-processor or Hyper-Threading Technology¹ capable systems.

Windows® Command	Linux® Command	Comment
/Qopenmp	-openmp	Enables the parallelizer to generate multithreaded code based on the OpenMP® directives.
/Qopenmp_report{0 1 2}	-openmp_report{0 1 2}	Controls the OpenMP parallelizer's diagnostic levels. The default is /Qopenmp_report1.
/Qparallel	-parallel	Detects parallel loops capable of being executed safely in parallel and automatically generates multithreaded code for these loops.
/Qpar_report{0 1 2 3}	-par_report{0 1 2 3}	Controls the auto-parallelizer's diagnostic levels as follows: 0 - displays no diagnostic information. 1 - indicates loops successfully parallelized (default). 2 - loops successfully and unsuccessfully parallelized. 3 - adds information about any proven or assumed dependencies inhibiting parallelization.
/Qpar_threshold[n]	-par_threshold[n]	Sets a threshold for the autoparallelization of loops based on the probability of profitable execution of the loop in parallel, n=0 to 100 . Default: n=75 . This option is used for loops whose computation work volume cannot be determined at compile time. 0 - parallelize loops regardless of computation work volume. 100 - parallelize loops only if profitable parallel execution is almost certain.

For product and purchase information visit:
www.intel.com/software/products

Intel, the Intel logo, Itanium, Pentium, Intel Centrino, Intel Xeon, Intel XScale, and VTune are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
¹Other names and brands may be claimed as the property of others.
¹Hyper-Threading Technology requires a computer system with an Intel® Pentium® 4 processor supporting HT Technology and a Hyper-Threading Technology enabled chipset, BIOS, and operating system. Performance will vary depending on the specific hardware and software you use. See <http://www.intel.com/info/hyperthreading> for more information including details on which processors support HT Technology.
Copyright © 2004, Intel Corporation. All Rights Reserved.
1004/AXB/ITF/PDF/XX 254349-003



Quick-Reference Guide to Optimization with Intel® Compilers

For Intel® Pentium® 4 and Itanium® processors and systems with Intel® Extended Memory 64 Technology (Intel® EM64T).

Application Performance

A Step-by-Step Approach to Application Tuning with Intel® Compilers

Before you begin performance tuning, ensure that your application runs as intended with a base set of options or in debug-mode (-Od and -Zi).

- 1 Use the Automatic Optimization Options (-O1, -O2, or -O3) and determine which one works best for your application by measuring performance with each.
- 2 Add in Interprocedural Optimization (IPO) and/or Profile-Guided Optimization (PGO) and again measure performance to determine if your application benefits from either of them.
- 3 Fine-tune performance with processor-specific options to target systems based on Pentium 4 and Itanium processors, including those with Intel EM64T technology. This step works best by identifying performance “hot-spots” with the Intel® VTune™ Performance Analyzer so you know which parts of your application need specific tuning. Also, the Intel compilers’ optimization reports show where the compiler could use more of your help.
- 4 Run your applications on multi-processor or Hyper-Threading Technology¹ capable systems using the Parallel Performance options.

Maximize
Application Performance
on Intel® architectures



Automatic Optimization Options

Before you begin performance tuning, ensure that your application runs as intended with a base set of options or in debug mode (-Od and -Zi). These are general optimization options that should be at the heart of any application tuning for all 32- and 64-bit Intel® processors. Try these different options and measure your performance before proceeding to more advanced optimizations.

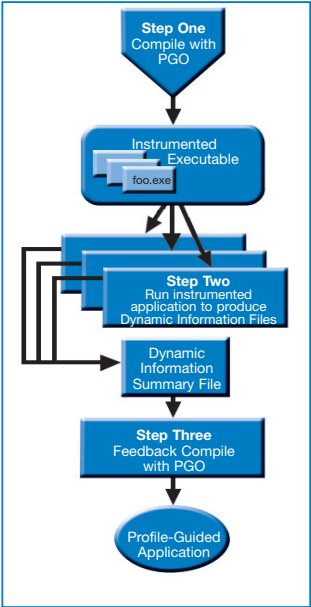
Windows® Command	Linux® Command	Comment
/Od (No Optimization)	-O0	No optimization. Useful during application development and debugging.
/O1 (Optimize for size)	-O1	Omits optimizations that tend to increase object size. Creates the smallest optimized code in most cases. On Linux systems with IA-32 processors only, there is no difference between -O1 and -O2 . This option has proven useful in many large server/database applications where memory paging due to larger code size is an issue.
/O2 (Maximize speed)	-O1 or -O2	Default setting. Creates the fastest code in most cases, but may increase code size significantly over /O1 . On Linux systems with IA-32 processors, -O1 and -O2 are equivalent.
/Ox (Maximize optimization)	n/a	Equivalent to /O2 except that /Ox does not imply /Gy (function packaging) or /Gf (string pooling).
/O3 (High-level optimizations)	-O3	Same as /O2 , plus loop transformations and data prefetching for improved memory usage efficiency. For the full benefit of /O3 on Intel 32-bit processors, also use the /Qx{K, W, N, B, P} or /Qax{K, W, N, B, P} options for Pentium® III and Pentium 4 processors and subsequent IA-32 processors. This option has proven useful for a broad range of applications, particularly loopy, kernel-based code common in high-performance computing.
/fast	-fast	The -fast option maximizes speed across the entire program. For Itanium-based systems, -fast sets -O3, -ipo, and -static. For IA-32 and systems using Intel® EM64T, -fast sets -O3, -ipo, -static, and -xP.
/Zi	-g	Generates debug information for use with any of the common platform debuggers.

Interprocedural Optimization (IPO) and Profile-Guided Optimization (PGO) Options

IPO controls function-inlining to reduce function call overhead and improve data layout across functions. PGO provides run-time feedback to guide optimization decisions about data and code layout to improve instruction-cache, paging and branch prediction. Additionally, IPO can increase code size. Be sure to measure your execution performance, compile-time, and code-size tradeoffs with these options. IPO is best used in conjunction with PGO to guide which functions to inline.

Windows® Command	Linux® Command	Comment
/Qip	-ip	Single file optimization. Allows selective inlining optimization within a single source file.
/Qipo[<i>value</i>]	-ipo[<i>value</i>]	Permits inlining and other optimizations among multiple source files. The optional value argument controls the maximum number of link-time compilations (or number of object files) that are spawned. The default for <i>value</i> is 1.
/Qipo_separate	-ipo_separate	Creates one object file for every source file. This option overrides -ipo[<i>value</i>] .
/Qprof_gen	-prof_gen	Instruments a program for profiling.
/Qprof_dirdir	-prof_dirdir	Specifies a directory for the profiling output files, *.dyn and *.dpi.
/Qprof_use	-prof_use	Enables use of profiling information during optimization.

Profile-Guided Optimization (PGO) Steps



Optimizations for Intel® Pentium® 4 Processors and Systems with Intel® EM64T

These options allow you to tune performance specifically for the Intel processor-based systems you are targeting. As with each previous step, measure the performance benefit of each option to guide your decisions. Use the Intel compilers’ optimization reports to assist in determining whether you can provide more help to the compiler in the form of anti-aliasing or memory disambiguation information.

Intel EM64T-Specific Optimization Recommendation: Use the **-QaxP** (-axP on Linux) for best performance on systems based on processors with Intel EM64T while still providing best performance on any Intel Pentium 4 processor or compatible system.

IA-32-Specific Optimization Recommendation: Use the **-QaxN** (-axN on Linux), for best performance across all Intel Pentium 4 processors and the Pentium M processor. (You may also want to experiment with **-QaxB** (-axB) on Pentium M processors.)

Windows® Command	Linux® Command	Comment
/Qax{K W N B P}	-ax{K W N B P}	Automatic Processor Dispatch. Generates specialized code for the indicated processors while also generating generic IA-32 code. You can use more than one code to tune for multiple processors in the same executable. K – Intel® Pentium® III and compatible Intel processors W - Intel Pentium 4 and compatible Intel processors, excluding those with Intel® EM64T B - Intel Pentium M and compatible Intel processors P - Intel Pentium 4 processor with Streaming SIMD Extensions 3 and compatible Intel processors N - provides additional Pentium 4 processor tuning beyond W . Only the -axW and -axP options are available for Intel EM64T.
/Qx{K W N B P}	-x{K W N B P}	Processor-specific Targeting. Generates specialized code for the indicated processor. The executable should only be run on the targeted compatible processors. K - Intel Pentium III and compatible Intel processors W - Intel Pentium 4 and compatible Intel processors, excluding those with Intel EM64T B - Intel Pentium M and compatible Intel processors P - Intel Pentium 4 processor with Streaming SIMD Extensions 3 and compatible Intel processors N - provides additional Pentium 4 processor tuning beyond W. N, B, and P generate a run-time check to determine that the correct compatible Intel processor is used to prevent potential run-time faults that could otherwise occur with K and W . Only the -xW and -xP options are available for Intel EM64T.
/Qprefetch[-]	-prefetch[-]	Enables or disables prefetch insertion (requires -O3).
/Qfp_port	-fp_port	Rounds floating-point results after floating-point operations, so rounding to user-declared precision happens at assignments and type conversions; this has some impact on speed. The default is to keep results of floating-point operations in higher precision. Use this if you are experiencing differences in floating-point precision versus other platforms.
/Qvec_report{0 1 2 3 4 5}	-vec_report{0 1 2 3 4 5}	Controls amount of vectorizer diagnostic information as follows: n = 0 : no information n = 1 : indicates vectorized loops (default) n = 2 : indicates vectorized and non-vectorized loops n = 3 : indicates vectorized and non-vectorized loops and prohibits data dependence information n = 4 : indicates non-vectorized loops n = 5 : indicates non-vectorized loops and prohibits data dependence information